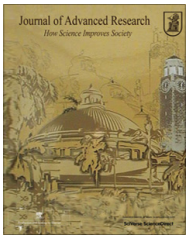
Journal of Advanced Research (2015) 6, 987–993

Cairo University

Journal of Advanced Research



ORIGINAL ARTICLE

Sort-Mid tasks scheduling algorithm in grid computing

Naglaa M. Reda [a], A. Tawfik [b], Mohamed A. Marzok [b,*], Soheir M. Khamis [a]

[a] Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt
[b] Egypt. Ctr. for Theo. Phys., Faculty of Engineering, Modern University, Cairo, Egypt

ARTICLE INFO

Article history:
Received 14 July 2014
Received in revised form 10 November 2014
Accepted 21 November 2014
Available online 26 November 2014

Keywords:
Grid computing
Heuristic algorithm
Scheduling
Resource utilization
Makespan

ABSTRACT

Scheduling tasks on heterogeneous resources distributed over a grid computing system is an NP-complete problem. The main aim for several researchers is to develop variant scheduling algorithms for achieving optimality, and they have shown a good performance for tasks scheduling regarding resources selection. However, using of the full power of resources is still a challenge. In this paper, a new heuristic algorithm called Sort-Mid is proposed. It aims to maximizing the utilization and minimizing the makespan. The new strategy of Sort-Mid algorithm is to find appropriate resources. The base step is to get the average value via sorting list of completion time of each task. Then, the maximum average is obtained. Finally, the task has the maximum average is allocated to the machine that has the minimum completion time. The allocated task is deleted and then, these steps are repeated until all tasks are allocated. Experimental tests show that the proposed algorithm outperforms almost other algorithms in terms of resources utilization and makespan.

Introduction

Grid computing systems [1,2] are distributed systems, enable large-scale resource sharing among millions of computer systems across a worldwide network such as the Internet. Grid resources are different from resources in conventional distributed computing systems by their dynamism, heterogeneity, and geographic distribution. The organization of the grid infrastructure involves four levels. First: the foundation level, it includes the physical components. Second: the middleware level, it is actually the software responsible for resource management, task execution, task scheduling, and security. Third: the services level, it provides vendors/users with efficient services. Fourth: the application level, it contains the services such as operational utilities and business tools.

The scheduling has become one of the major research objectives, since it directly influences the performance of grid applications. Task scheduling [3] is the main step of grid resource management. It manages jobs to allocate appropriate resources by using scheduling algorithms and polices. In static scheduling, the information regarding all the resources as well as all the tasks is assumed to be known in advance, by the time the application is scheduled. Furthermore, each task is assigned once to a resource. While in dynamic scheduling, the task allocation is done on the go as the application executes, where it is not possible to find the execution time. Tasks are entering dynamically and the scheduler has to work hard in decision making to allocate resources. The advantage of the dynamic over the static scheduling is that the system does not need to posse the run time behavior of the application before it runs.

* Corresponding author. Tel.: +20 1066286275.
E-mail address: m_sayed85@yahoo.com (M.A. Marzok).

Peer review under responsibility of Cairo University.

ELSEVIER Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jare.2014.11.010

Since the late nineties, several heuristic algorithms for grid task scheduling (GTS) [4–6] have been developed to improve grid performance. They are classified into task algorithms in which all tasks can be run independently and DAG algorithms, where a DAG represents the partial ordering dependence relation between tasks execution.

The main contribution of this work is to introduce an efficient heuristic algorithm for scheduling tasks to resources on computational grids with maximum utilization and minimum makespan. The proposed algorithm (Sort-Mid) depends on the minimum completion time and the average value AV of completion times for each task. It puts constrains to map the most appropriate task to the best convenient resource, which increases the grid efficiency. Performance tests show a good improvement over existing popular scheduling algorithms.

The most popular task GTS algorithms are surveyed in the following subsection. The rest of this paper is organized as follows. The proposed methodology with the suggested algorithm for the scheduling problem in grid computing system is introduced. Then, the used experimental materials followed by results and discussion are presented. Finally, the conclusion of the overall work is given.

Related work

Opportunistic load balancing (OLB) algorithm [7] assigns each task in arbitrary order to the next available machine regardless of the task's expected execution time on the machine, while, minimum execution time (MET) algorithm [8] assigns each task in arbitrary order to the machine with the minimum execution time without considering resource availability. But, minimum completion time (MCT) algorithm [8] assigns each task in arbitrary order to the machine with the earliest completion time. On the other hand, Min-Min algorithm [9,10] selects the machine with minimum expected completion time and assigns task with the MCT to it. Where, Max-Min algorithm [9,10] selects the machine with minimum expected completion time and the task with the maximum completion time is mapped to it. And, switching algorithm (SA) [9] combines MCT and MET to overcome some limitations of both methods.

Furthermore, Suffrage heuristic [9] maps the machine to the task that would suffer most in terms of expected completion time according to an evaluated suffrage value. Switcher heuristic [11] switches between the Max-Min and Min-Min algorithms by taking a scheduling decision based on the standard deviation of minimum completion time of unassigned jobs. RASA heuristic [12] has built a matrix representing the completion time of each task on every resource, and applies Min-Min if the number of available resources is odd, otherwise it applies Max-Min. Min-mean heuristic [13] reschedules the Min-Min produced schedule by considering the mean makespan of all the resources. Load balanced Min-Min (LBMM) heuristic [14] has reduced the makespan and has increased the resource utilization by choosing the resources with heavy load and reassigns them to the resources with light load. Mact-mini heuristic [15] maps the task with the maximum average completion time to the machine with minimum completion time. Recently, a new heuristic algorithm based on Min-Min has been presented [16]. It selected resources according to a new makespan value and the maximum value of possibilities tasks (MVPT).

Methodology

In this section, we present a new idea for solving the scheduling problem in a grid. Scheduling is the main step of grid machines management [17]. Machines may be homogeneous or heterogeneous. A grid scheduler selects the best machine to a particular job and submits that job to the selected machine [18]. The main aim of suggested heuristic algorithm for scheduling a set of tasks on a computational grid system is to maximize the machines utilization and to minimize the makespan. Given a grid G with a finite number, m, of machines (resources); M_1, $M_2, \ldots, M_m$, $m > 1$. Let T be a finite nonempty set of n tasks; T_1, $T_2, \ldots, T_n$, $n > 1$ that needs to be executed in G.

In the following work, the proposed algorithm called Sort-Mid is given. It's steps to assign each task to a suitable machine are summarized below. It uses assignment function S: $T \rightarrow G$ which is defined as follows. For every positive integer $i \leqslant n$; $\exists$ a positive integer $j \leqslant m$ s.t. $S(T_i) = M_j$. The first step is to sort the completion times (SCT) of each task T_i in T in increasing order. The introduced scheduling decision is based on computing the average value AV of two consecutive completion times in SCT for each T_i. AV is computed by $(SCT_K + SCT_{K+1})/2$, where $k = \lceil m/2 \rceil$. In the second step, the task having the maximum AV is selected. In the third step, the task is assigned to the machine possessing minimum completion time. Next, the assigned task is deleted from T. Finally, the waiting time for the machine that executes this task is updated. These steps are repeated until all n tasks are scheduled on m machines. The pseudo code of the algorithm is as listed below.

Algorithm Sort-Mid:

Input: Number of tasks n, Number of machines m, Grid $G = \{M_1, M_2, \ldots, M_m\}$, Tasks $T = \{T_1, T_2, \ldots, T_n\}$, Machines availability R; Estimated time of computation ETC.
Output: The result of the assignment function S: $S(T_1), S(T_2), \ldots, S(T_n)$.
Begin
Initialization: $A \leftarrow \{1,2,\ldots,n\}$, $K \leftarrow \lceil m/2 \rceil$, $CT \leftarrow ETC$;
1. **While** $A \neq \emptyset$ **do**
2. **If** $|A| \neq 1$ **Then**
3. $Max\_value \leftarrow 0$, $Index\_machine \leftarrow 0$, $Index\_task \leftarrow 0$;
4. **For** all $i \in A$ **do**
5. $SCT \leftarrow$ sort CT [i] in ascending order;
6. $AV \leftarrow (SCT_K + SCT_{K+1}) / 2$;
7. **If** $AV > Max\_value$ **Then**
8. $Max\_value \leftarrow AV$;
9. $Index\_task \leftarrow i$;
10. $Index\_machine \leftarrow$ index of machine whose completion time equals SCT_1;
11. **End If**
12. **End For**
13. $S(T_{Index\_task}) \leftarrow M_{Index\_machine}$;
14. $A \leftarrow A - \{Index\_task\}$;
15. $R_{Index\_machine}(R_{Index\_machine} + ECT_{Index\_task,Index\_machine}$;
16. **For** all $i \in A$ **do**
17. $CT_{i,Index\_machine} \leftarrow ECT_{i,Index\_machine} + R_{Index\_machine}$;
18. **End For**
19. **Else**
Assign the remaining task to the machine having the minimum completion time and delete it;
20. Update waiting time of machine executing it;
21. **End If**
22. **End While**
End.

It is clear that Sort-Mid algorithm is correct, since at the end, the set of tasks indices are vanished, i.e., all tasks are assigned to appropriate machines.

In the following, we analyze the time complexity of the above given algorithm.

Lemma. *The time complexity of Algorithm Sort-Mid is in $O(n^2 m \log n)$, where n and m are the numbers of tasks and machines in a grid computing system, respectively.*

Proof. It is obvious that the first For-loop starting from step 4 to step 12 iterates n time. Each iteration costs at least ($m \log m$), which is one run of step 5 to sort the elements at row number i of CT in an ascending order. Also, the second For-loop starting from step 16 to step 18 which updates the wait time, costs O (n). □

And, one run to select task and delete it and update time take $n + m + n$, respectively. Since the while-loop (Starting from step 1 to step 22) executes n time, each run of them costs of ($nm \log m + 2n + m$). This implies that the total time complexity of the algorithm is in $O(n^2 m \log n)$.

An illustrative example

To clarify how the proposed algorithm Sort-Mid schedules tasks perfectly, consider the following example for a grid environment with three machines and three tasks. Its *ETC* matrix with special form is given in Fig. 1.

The initialization step initializes the CT by ETC and machines availability vector R by zeros.

At first iteration, Max_value = Index_machine = Index_task = 0 and $A = \{1,2,3\}$, then the number of elements $|A| = 3$, and the created SCT after sorting illustrates as follows.

$$\mathrm{SCT} = \begin{bmatrix} M_3:22 & M_2:23 & M_1:45 \\ M_3:22 & M_1:45 & M_2:70 \\ M_3:23 & M_1:25 & M_2:63 \end{bmatrix}$$

For the first row of SCT, the average value (AV) of the first task is AV (1) = ($\mathrm{SCT}_2 + \mathrm{SCT}_3$)/2 = (23 + 45)/2. After that, the new value 34 is compared with the value of Max_value = 0, so the Max_value = 34, Index_task = 1 and Index_machine = 3.

For the second row, the average value is AV (2) = ($\mathrm{SCT}_2 + \mathrm{SCT}_3$)/2 = (45 + 70)/2. Then after comparison, Max_value = 57.5, hence Index_task = 2 and Index_machine = 3.

For the third row, the average value AV (3) = ($\mathrm{SCT}_2 + \mathrm{SCT}_3$)/2 = (25 + 63)/2. And, the values of Max_value are still maximum value, Index_task and Index_machine will not change.

At the end of the first iteration, the task having Index_task (= 2) is deleted from the set A, then $A = \{1,3\}$. And $R_3 = 0 + 22 = 22$. And so, CT updates to the following matrix:

$$\mathrm{CT} = \begin{bmatrix} M_1:45 & M_2:23 & M_3:44 \\ M_1:25 & M_2:63 & M_3:45 \end{bmatrix}$$

At the second iteration for while-loop, first put Max_value = Index_machine = Index_task = 0, $A = \{1,3\}$ and $|A| = 2$. Then, SCT is arranged as follows:

$$\begin{matrix} t_1 \\ t_2 \\ t_3 \end{matrix} \begin{bmatrix} M_1:45 & M_2:23 & M_3:22 \\ M_1:45 & M_2:70 & M_3:22 \\ M_1:25 & M_2:63 & M_3:23 \end{bmatrix}$$

Fig. 1 The matrix *ETC* of the given.

Table 1 A comparison between algorithms in makespan and tasks scheduling.

Algorithms	M1	M2	M3	Makespan
MET			T_1, T_2, T_3	67
OLB	T_1	T_2	T_3	70
MCT	T_3		T_1, T_2	44
Max-Min		T_1	T_2, T_3	45
Min-Min	T_3		T_1, T_2	44
Sort-Mid	T_3	T_1	T_2	25

$$\mathrm{SCT} = \begin{bmatrix} M_2:23 & M_3:44 & M_1:45 \\ M_1:25 & M_3:45 & M_2:63 \end{bmatrix}$$

For the first row, the average value of the first task is AV (1) = ($\mathrm{SCT}_2 + \mathrm{SCT}_3$)/2 = (44 + 45)/2 in SCT. After comparing 44.5 with 0, then Max_value = 34, Index_task = 1 and Index_machine = 2.

For the second row, AV (3) = ($\mathrm{SCT}_2 + \mathrm{SCT}_3$)/2 = (45 + 63)/2. Then Max_value = 54, Index_task = 3 and Index_machine = 1.

At the end of second iteration, the task with index Index_task (= 3) is deleted and $A = \{1\}$, $R_1 = 0 + 25 = 25$, CT

$$\mathrm{CT} = [M_1:70 \quad M_2:23 \quad M_3:44]$$

In the third iteration, $A = \{1\}$ and $|A| = 1$, so the task with index 1 is assigned to the machine having the minimum completion time M_2. i.e., Index_task = 1 and Index_machine = 2.

Finally, at the end of third iteration, the remaining task is deleted, then $A = \emptyset$. And $R_2 = 0 + 23 = 23$.

As a result of the above execution, the makespan for the above example equals Max (22, 25 and 23) = 25. The makespan produced by other previous algorithms compared to the result of Sort-Mid algorithm is shown in Table 1.

Experimental materials

For comparison of our proposed heuristic with other scheduling algorithm, various heuristic algorithms have been developed to compare with Sort-Mid algorithm. In this section, the benchmark description is given, and the ETC model used as in benchmark experiments [14–20] is specified.

In this paper, we used the benchmark model [4]. The simulation model is based on expected time to compute (ETC) matrix for 512 tasks and 16 machines. An *ETC* matrix is said to be *consistent* (C) if whenever a machine m_j executes any task t_i faster than machine m_k, then machine m_j executes all tasks faster than machine m_k. In contrast, *inconsistent* matrices (I) characterize the situation where machine m_j may be faster than machine m_k for some tasks and slower for others. *Semi-consistent* matrices (S) happen when some machines are consistent while others are inconsistent. Also, different *ETC* matrix task and machine heterogeneity are studied, each one has two cases

Table 2 The ETC model.

Consistency	Heterogeneity			
	Task (High)		Task (Low)	
	Machine		Machine	
	High	Low	High	Low
Consistent	C_hihi	C_hilo	C_lohi	C_lolo
Inconsistent	I_hihi	I_hilo	I_lohi	I_lolo
Semi-consistent	S_hihi	S_hilo	S_lohi	S_lolo

high (hi) or low (lo). Thus, the twelve matrices are tested and abbreviated as shown in Table 2.

In addition, a computer program in VB language is developed for seven existing and proposed heuristic methods mentioned above. This program produces a schedule that maps tasks to available resources and calculates the objectives based on the *ETC* matrix supplied to it. The twelve different *ETC* matrices suggested by Braun et al. [4] for different scenarios mentioned in Table 2 are used as inputs to the computer program, and the results are analyzed in the following section.

Results and discussion

There are several performance metrics to evaluate the quality of a scheduling algorithm [3]. This section tests Sort-Mid algorithm mentioned in Section 'Methodology' according to these criteria. It considers the problem of scheduling n tasks on a heterogeneous grid system of m machines. It presents in the following a comparison of most recent and efficient algorithms against Sort-Mid in regard to each criterion for emphasizing its strength. In the following, we compare our heuristic algorithm with other scheduling algorithms via using benchmark experiments [4,19].

Computational complexity

The complexity is an essential metric in theoretical analysis of algorithms that asymptotically estimate their performance. It determines the amount of time to solve the given computational problem using selected mathematical notation such as the Big O. In our case, it indicates how fast the scheduling algorithm will be in finding a feasible solution in a highly dynamic heterogeneous grid system. Table 3 illustrated the complexity of Sort-Mid algorithm and other important ones. It is worth to remark that the number of machines in a grid m is much less than the number of tasks n and so log m. Therefore, in practical, the running time of Sort-Mid algorithm is approximately equal to the running time of Max-Min, Min-mean, Min-Min and suffrage algorithm.

Resource utilization

The grid's resource utilization is the most essential performance metric for grid managers. *The Machine's Utilization (MU)* is defined as the amount of time at which a machine is busy in executing tasks, while the *grid's resource utilization (GU)* is the average of machines' utilization. They are computed as follows:

$$GU = \frac{\sum_{j=1}^{m} MU_j}{m}$$

where,

$$MU_j = \frac{r_j}{makespan}, \quad for\ j = 1, 2, \ldots, m.$$

Fig. 2 and Tables 4–6 show the values of GUs for the eight mentioned algorithms. Sort-Mid gives the second maximum resource utilization for ten instances and third maximum resource utilization for two instances. The Max-Min gives the highest maximum resource utilization for all instances but the difference is very small, while the computed makespan of Sort-Mid algorithm is better than that of Max-Min in all instances.

Makespan

The makespan is an important performance criterion of scheduling heuristics in grid computing systems. It is

Table 3 Complexity comparison for Sort-Mid with other algorithms.

Algorithm	MET	OLB	Mact-min	Max-min	Min-mean	Min-Min	Suffrage	Sort-Mid
Complexity	O(nm)	O(nm)	O(nm)	O(n^2m)	O(n^2m)	O(n^2m)	O(n^2m)	$n^2m \log m$

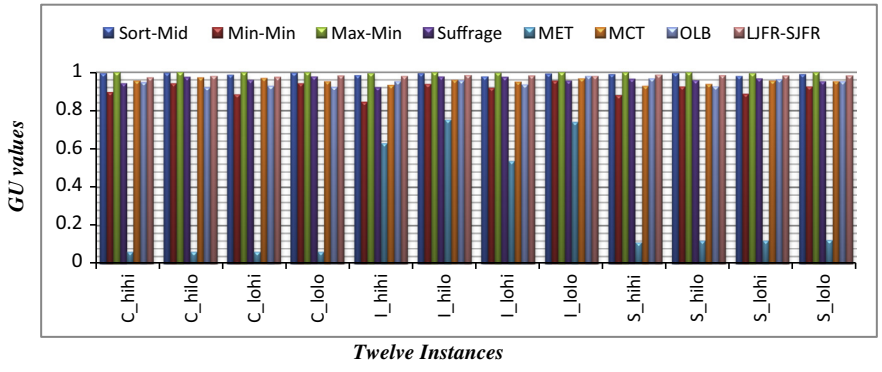


Fig. 2 A comparison of the GU values for 12 instances.

Table 4 Grid's resource utilization (consistent instance).

	C_hihi	C_hilo	C_lohi	C_lolo
Sort-Mid	0.99432	0.99542	0.9853	0.996616
Min-Min	0.89648	0.94337	0.8823	0.941213
Max-min	0.99882	0.99950	0.9989	0.999504
Suffrage	0.94325	0.97425	0.9595	0.976099
MET	0.0625	0.0625	0.0625	0.0625
MCT	0.95386	0.97069	0.9690	0.95151
OLB	0.94671	0.92035	0.9285	0.923213
LJFR-SJFR	0.96715	0.97862	0.9728	0.980576

Table 5 Grid's resource utilization (inconsistent instance).

	I_hihi	I_hilo	I_lohi	I_lolo
Sort-Mid	0.98453	0.99326	0.9756	0.991719
Min-Min	0.84491	0.93694	0.9179	0.953698
Max-Min	0.99367	0.99819	0.9959	0.998497
Suffrage	0.91986	0.97672	0.9744	0.955264
MET	0.62863	0.75058	0.5366	0.740376
MCT	0.93287	0.95978	0.9496	0.965665
OLB	0.95119	0.95589	0.9340	0.979608
LJFR-SJFR	0.97782	0.98267	0.9819	0.978605

Table 6 Grid's resource utilization (semi-consistent instance).

	S_hihi	S_hilo	S_lohi	S_lolo
Sort-Mid	0.9874	0.9941	0.9799	0.9888
Min-Min	0.87799	0.92539	0.8868	0.924657
Max-Min	0.99875	0.99917	0.9938	0.999145
Suffrage	0.96339	0.95712	0.9663	0.951159
MET	0.11088	0.12048	0.1219	0.124449
MCT	0.92827	0.93834	0.9539	0.951856
OLB	0.96709	0.92459	0.962	0.951005
LJFR-SJFR	0.98550	0.98387	0.9824	0.981659

Table 7 Makespan values of high task, high machine heterogeneity in case of C, I, and S benchmark models, respectively.

	C_hihi	I_hihi	S_hihi
Sort-Mid	9683148.7	3724452.312	5632995.76
Min-Min	9037587.109	4024444.672	5377382.055
Max-Min	12255384.79	7146473.427	9213627.859
Suffrage	11990851.28	4809887.958	7442261.93
MET	47472299.43	4508506.792	25162058.14
MCT	11422624.49	4413582.982	6693923.896
OLB	14376662.18	26102017.62	19464875.91
LJFR-SJFR	12368381.53	6129579.87	8295806.53

defined as the maximum completion time of application tasks executed on grid resources. Formally, it is computed by using the following equation. Note that C is the matrix

Table 8 Makespan values of high task, low machine heterogeneity in case of C, I, and S benchmark models, respectively.

	C_hilo	I_hilo	S_hilo
Sort-Mid	175920.6628	87965.99592	116233.14
Min-Min	166828.8663	83379.01434	110333.114
Max-Min	207680.683	143476.485	167058.1754
Suffrage	188756.6255	99838.9465	136540.7513
MET	1185092.969	96610.48102	605363.7727
MCT	185887.4041	94855.91348	126587.5914
OLB	221051.8236	272785.2008	250362.1138
LJFR-SJFR	200846.4618	128909.6339	153719.3364

Table 9 Makespan values of low task, high machine heterogeneity in case of C, I, and S benchmark models, respectively.

	C_lohi	I_lohi	S_lohi
Sort-Mid	325366.0837	134330.3048	169284.5168
Min-Min	291711.0926	124644.635	153307.7354
Max-min	398822.906	255370.7475	272001.9739
Suffrage	397193.1733	140382.7428	179748.7113
MET	1453098.004	185694.5945	674689.5356
MCT	378303.6246	143816.0937	186151.2863
OLB	477357.0195	833605.6545	603231.4673
LJFR-SJFR	390605.4791	212557.6419	246246.4265

Table 10 Makespan values of low task, low machine heterogeneity in case of C, I, and S benchmark models, respectively.

	C_lolo	I_lolo	S_lolo
Sort-Mid	5884.438158	3041.923489	4008.148616
Min-Min	5670.939533	2835.886811	3943.347953
Max-min	7020.853442	4967.738767	6176.0686
Suffrage	6052.637899	2947.256655	4081.267844
MET	39582.29732	3399.284768	21042.41343
MCT	6360.054945	3137.350329	4436.117532
OLB	7306.595595	8938.026908	8938.389213
LJFR-SJFR	6767.322547	4321.483534	5584.607333

of the completion times after executing given tasks in grid computing system and R is the vector of waiting times of m machines.

$$Makespan = max\{c_{ij} | \forall 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}, \quad or$$

$$Makespan = max\{r_j | \forall 1 \leqslant j \leqslant m\}.$$

The makespan of the scheduling algorithms for the twelve different instances of the ETC matrices is shown in Tables 7–10. Furthermore, Fig. 3 illustrates a comparison of the makespan between Sort-Mid and other algorithms for the above case study. In addition, Table 11 gives the rank of all heuristics based on grid's resources utilization and makespan value of respective schedule for different instances.

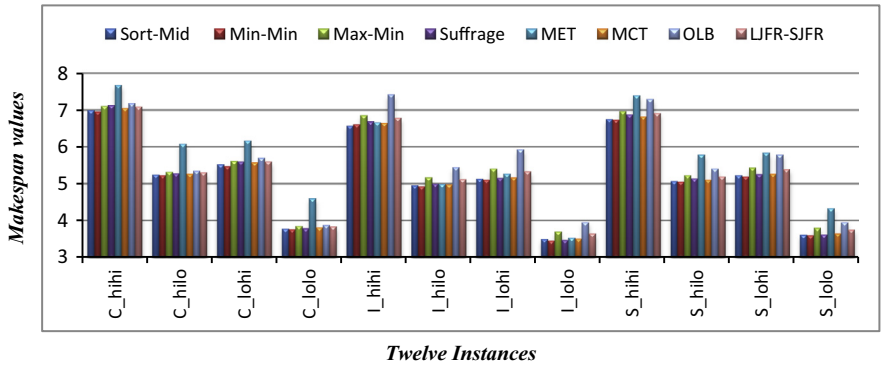


Fig. 3 Makespan comparison for 12 instances.

Table 11 Rank of heuristics based on resources utilization and makespan.

	Grid's resources utilization			Makespan		
	I	II	III	I	II	II
C_hihi	Max-Min	Sort-Mid	LJFR-SJFR	Min-Min	Sort-Mid	MCT
C_hilo	Max-Min	Sort-Mid	LJFR-SJFR	Min-Min	Sort-Mid	MCT
C_lohi	Max-Min	Sort-Mid	LJFR-SJFR	Min-Min	Sort-Mid	MCT
C_lolo	Max-Min	Sort-Mid	LJFR-SJFR	Min-Min	Sort-Mid	Suffrage
I_hihi	Max-Min	Sort-Mid	LJFR-SJFR	Sort-Mid	Min-Min	MCT
I_hilo	Max-Min	Sort-Mid	LJFR-SJFR	Min-Min	Sort-Mid	MCT
I_lohi	Max-Min	LJFR-SJFR	Sort-Mid	Min-Min	Sort-Mid	Suffrage
I_lolo	Max-Min	Sort-Mid	OLB	Min-Min	Suffrage	Sort-Mid
S_hihi	Max-Min	Sort-Mid	LJFR-SJFR	Min-Min	Sort-Mid	MCT
S_hilo	Max-Min	Sort-Mid	LJFR-SJFR	Min-Min	Sort-Mid	MCT
S_lohi	Max-Min	LJFR-SJFR	Sort-Mid	Min-Min	Sort-Mid	Suffrage
S_lolo	Max-Min	Sort-Mid	LJFR-SJFR	Min-Min	Sort-Mid	Suffrage

Conclusions

Selecting the appropriate resource for a specific task is one of the challenging work in computational grid. This work introduces a new task scheduling algorithm called Sort-Mid. The implementation of Sort-Mid algorithm and various existing algorithms are tested using a benchmark simulation model. Min-Min is the simplest and common scheduling algorithm for grid computing. But, it works poorly when the number of large tasks is less than the number of small tasks. Also, the computed makespan by Min-Min in this case is not good. The computed grid's resources utilization by Min-Min is not good. To avoid the disadvantages of grid's resources utilization and makespan, Sort-Mid is designed to maximize grid's resources utilization and to minimize the makespan. This algorithm overcomes the affection of large varies of task's execution times. A comparison of makespan values between our algorithm and other seven scheduling algorithm has been conducted. Obviously, the result of Sort-Mid is better than all algorithms in the eleven underling instances except for Min-Min. Nevertheless, Sort-Mid is the best in case of inconsistent high task and high machine heterogeneity. On the other hand, experimental results indicate that Sort-Mid utilizes the grid by more than 99% at 6 instances and more than 98% at 4 instances.

In conclusion, the rank of the proposed Sort-Mid algorithm regarding both makespan and utilization is very good.

Conflict of Interest

The authors have declared no conflict of interest.

Compliance with Ethics Requirements

This article does not contain any studies with human or animal subjects.

References

[1] Magoulès F, Pan J, Tan K, Kumar A. Introduction to grid computing. London, New York: CRC Press; 2009.
[2] Amalarethinam, George DI, Muthulakshmi P. An overview of the scheduling policies and algorithms in grid computing. Int J Res Rev Comput Sci 2011;2(2):280–94.
[3] Chandak A, Sahoo B, Turuk A. An overview of task scheduling and performance metrics in grid computing. Int J Comput Appl 2011:30–3.
[4] Braun TD, Siegel HJ, Beck N, Boloni LL, Maheswaran M, Reuther AL, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous

distributed computing systems. J Parallel Distrib Comput 2011;61(6):810–37.
[5] Ma T, Yan Q, Liu W, Guan D, Lee S. Grid task scheduling: algorithm review. IETE Tech Rev 2011;28(2):158–67.
[6] Elzeki OM, Rashad MZ, Elsoud MA. Overview of scheduling tasks in distributed computing systems. Int J Soft Comput Eng 2012;2(3):470–5.
[7] Maheswaran M, Ali S, Siegel HJ, Hensgen D, Richard FF. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. HCW '99, DC, USA; 1999. p. 30–44.
[8] Freund RF, Gherrity M, Ambrosius S, Campbell M, Halderman M, Hensgen D, et al. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. HCW '98, Orlando, FL; 1998. p. 184–99.
[9] Maheswaran M, Ali S, Siegel HJ, Hensgen D, Richard FF. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J Parallel Distrib Comput 1999;59(2):107–31.
[10] Anousha S, Ahmadi M. An improved Min-Min task scheduling algorithm in grid computing. LNCS 7861; 2013. p. 103–13.
[11] Singh M, Suri PK. $QPS_{Max\text{-}Min<>Min\text{-}Min}$: a QoS based predictive Max-Min, Min-Min switcher algorithm for job scheduling in a grid. Inform Technol J 2008;7(8):1176–81.
[12] Parsa S, Entezari-Maleki R. RASA: a new grid task scheduling algorithm. Int J Digital Content Technol Appl 2009;3(4):91–9.
[13] Kamalam GK, Bhaskaran V. A new heuristic approach: Min-Mean algorithm for scheduling meta-tasks on heterogeneous computing systems. Int J Comput Sci Network Secur 2010;10(1):24–31.
[14] Kokilavani T, Amalarethinam DI. Load balanced Min-Min algorithm for static meta-task scheduling in grid computing. Int J Comput Appl 2011;20(2):43–9.
[15] Alharbi F. Multi objectives heuristic algorithm for grid computing. Int J Comput Appl 2012;46(18):39–45.
[16] Anousha S, Anousha S, Ahmadi M. A new heuristic algorithm for improving total completion time in grid computing. Lect Notes Electr Eng 2014;308:17–26.
[17] Chaturvedi AK, Sahu R. New heuristic for scheduling of independent tasks in computational grid. Int J Grid Distrib Comput 2011;4(3):25–36.
[18] Kousalya K, Balasubramanie P. Ant algorithm for grid scheduling powered by local search. Int J Open Probl Compt Math 2008;1(3):222–40.
[19] Braun TD, Siegel HJ, Beck N, Boloni LL, Maheswaran M, Reuther AL, et al. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. HCW'99, DC, USA; 1999. p. 15–29.
[20] Merajiand S, Salehnamadi MR. A batch mode scheduling algorithm for grid computing. J Basic Appl Sci Res 2013;3(4):173–81.